Available online at www.sciencedirect.com
SCIENCE DIRECT®

Discrete Applied Mathematics 145 (2004) 80–94

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

Jackson's pseudo-preemptive schedule and cumulative scheduling problems

Jacques Carlier[a], Eric Pinson[b]

[a] Laboratoire HEUDIASYC UMR CNRS 6599, Université de Technologie de Compiègne, France
[b] Centre de Recherches et d'Etudes sur l'Application des Mathématiques, Institut de Mathématiques Appliquées, Université Catholique de l'Ouest, Angers, France

Received 8 January 2001; received in revised form 12 September 2002; accepted 19 September 2003

Abstract

The aim of this paper is to show the usefulness of the Jackson's pseudo-preemptive schedule (JPPS) for solving cumulative scheduling problems. JPPS was introduced for the m-processor scheduling problem $\mathrm{Pm}/r_i, q_i/C_{\max}$. In the latter problem, a set I of n operations has to be scheduled without preemption on m identical processors in order to minimize the makespan. Each operation i has a release date (or head) r_i, a processing time p_i , and a tail q_i. In the cumulative scheduling problem (CuSP), an operation i requires a constant amount e_i of processors throughout its processing. A CuSP is obtained, for instance, from the resource constrained project scheduling problem (RCPSP) by choosing a resource and relaxing the constraints induced by the other resources. We state new properties on JPPS and we show that it can be used for studying the CuSP and for performing adjustments of heads and tails using a strategy very close to the one designed by Carlier and Pinson for the $1/r_i$, $q_i/C_{\max}$ sequencing problem. It confirms the interest of JPPS for solving RCPSP.

Keywords: Scheduling; Resource; Adjustments; Heads; Makespan; Tails

1. Introduction

The aim of this paper is to demonstrate that the Jackson's pseudo-preemptive schedule (JPPS) introduced in [12] could become a very powerful tool for solving cumulative scheduling problems. JPPS is a generalization of Jackson's preemptive schedule (JPS) to the case where more than one unit of resource is available. Our results confirm the robustness of Jackson's rule [18] which has been defined 50 years ago for the one machine sequencing problem. Jackson's rule consists in sequencing the operations in an ascending order of their due dates. It minimizes maximal lateness when all operations are available at the same time. In the sequel, due dates are replaced by tails. So, instead of minimizing maximum lateness, we focus on the makespan minimization. We will also introduce release dates. For instance, in the one processor sequencing problem $1/r_i$, $q_i/C_{\max}$, a set I of n operations has to be scheduled without preemption on a single processor in order to minimize the makespan. Each operation $i \in I$ has a release date or head r_i , a processing time p_i, and a latency duration or tail q_i. $1/r_i$, $q_i/C_{\max}$ is NP-hard in the strong sense, but it is at the borderline of easy and hard problems. Indeed, it is not so difficult to solve in practice [5,20]. This can be explained by studying its preemptive version $1/r_i$, q_i, $\mathrm{pmtn}/C_{\max}$. The latter is solved by computing the list schedule associated with Jackson's rule, so called JPS. JPS makespan is a very tight lower bound for the $1/r_i$, $q_i/C_{\max}$ problem. Indeed, the difference between the makespans of Jackson's schedules in the preemptive and non-preeemptive cases is smaller than $p_{\max}$ (maximal processing time over the set of tasks) [5]. Moreover, JPS can be computed in $\mathrm{O}(n \log n)$ time, and has a very nice structure which can be intensively exploited for

E-mail addresses: carlier@utc.fr (J. Carlier), pinson@ima.uco.fr (E. Pinson).

0166-218X/$ - see front matter
doi:10.1016/j.dam.2003.09.009

solving NP-hard disjunctive scheduling problems. For instance, it permits efficient adjustments of heads and tails and thus becomes an essential tool for solving the job shop problem [1,3,4,9–11,24].

The m parallel and identical processor scheduling problem $\mathrm{P}m/r_i,\ q_i/C_{\max}$ is a generalization of $1/r_i,\ q_i/C_{\max}$ where m identical processors are available for processing the operations. $\mathrm{P}m/r_i,\ q_i/C_{\max}$ is also NP-hard in the strong sense, but it seems harder to solve in practice than $1/r_i, q_i/C_{\max}$ [7,16]. Its preemptive version can be solved with an $\mathrm{O}(n^3(\log n + \log p_{\max}))$-time complexity algorithm, where $p_{\max} = \max_{i \in I} p_i$ [17,19]. But such a complexity forbids its intensive use in an enumerative process. Recently, we have introduced the JPPS for the m parallel and identical processor scheduling problem [12]. JPPS generalizes the JPS defined for the one processor sequencing problem. In a JPPS, we allow an operation to be processed on more than one processor at a time. For building it, we use a list algorithm whose priority dispatching rule is the complete tail, i.e. priority is given to the operations with maximal $q_i + a_i(t)$, where $a_i(t)$ is the remaining processing time of operation i at the current time t in the list algorithm. Notice that JPPS is not a valid preemptive schedule in the sense that it allows the use of a rational number of processors at the same time for an operation. Muntz and Coffman [13] associated also an instantaneous rate with an operation, but this rate is supposed to be smaller than or equal to 1. They proved that, because of the latter condition, a preemptive schedule can be built using Mac Naughton algorithm on each time interval in which rates are constant. Lawler [19,21] presented an extension of list schedules in the preemptive context (priority schedules) generalizing the idea of Coffman and Muntz. Next, Liu and Sanlaville [22,25] proposed the smallest laxity first (SLF) rule which is strictly equivalent to the complete tail rule (CTR) used for designing JPPS. SLF rule is not optimal for preemptive problem with release dates, hence it cannot be used for computing a lower bound for the non-preemptive problem. The makespan of JPPS can be computed in $\mathrm{O}(n \log n + nm \log m)$ time and is a tight lower bound for the $\mathrm{P}m/r_i,\ q_i/C_{\max}$ scheduling problem. An interesting property of JPPS is that its gap to an optimal non-preemptive solution is smaller than $2.p_{\max}$ [7].

The cumulative scheduling problem (CuSP) [2] generalizes itself $\mathrm{P}m/r_i,\ q_i/C_{\max}$ problem. In a CuSP, each activity requires a constant amount e_i of processors throughout its processing. This problem is of prime interest for solving more complex scheduling problems like the resource constrained project scheduling problem (RCPSP) [14,15,23]. Indeed, a CuSP can be obtained from a RCPSP instance by selecting a particular resource and relaxing the constraints induced by all the remaining resources.

The aim of this paper is to study the structure of JPPS in pointing out its numerous similarities with JPS. In particular, an efficient $\mathrm{O}(n^2)$ algorithm computing JPPS is proposed. We show that adjustments of heads and tails can be performed for the $\mathrm{P}m/r_i,\ q_i/C_{\max}$ scheduling problem by using a strategy very close to the one designed by Carlier and Pinson for the $1/r_i,\ q_i/C_{\max}$ scheduling problem. Moreover, we propose a simple adaptation of these tools (lower bounds and adjustments of heads) for the CuSP without any additional computational effort. Consequently, JPPS provides elimination rules and lower bounds for the RCPSP.

The paper is organized as follows. In Section 2, the JPPS principle and its basic theoretical properties are recalled. In Section 3, we state new results related to the structure of JPPS. Next, in Section 4, we explain how JPPS can be efficiently used for adjusting heads for the $\mathrm{P}m/r_i,\ q_i/C_{\max}$ scheduling problem. Section 5 deals with simple adaptations of the previous methods to the CuSP. Last, in Section 6, we conclude the paper by pointing out further research directions.

2. The JPS and the JPPS: some recalls

2.1. Introduction

This section deals with a brief presentation of the ideas underlying the design of JPS and JPPS defined, respectively, for the $1/r_i,\ q_i/C_{\max}$ and $\mathrm{P}m/r_i,\ q_i/C_{\max}$ scheduling problems. Both preemptive schedules are recalled and illustrated with simple examples.

2.2. JPS

JPS is the list schedule associated with the most work remaining (MWR) priority dispatching rule [18]. To build JPS, we schedule, at the first moment t where the processor and at least one operation are available, the available operation with maximal tail (an operation i is available at t if $r_i \leqslant t$ and if it is not completed at t). This operation is processed either up to its completion, or until a more urgent operation becomes available. We update t and iterate until all the operations are scheduled. By using heap structures, JPS can be computed in $\mathrm{O}(n \log n)$ time. Its makespan is equal to $\max_{J \subseteq I} h(J)$, where $h(J) = \min_{j \in J} r_j + \sum_{j \in J} p_j + \min_{j \in J} q_j$ [5]. Fig. 1 gives an example of JPS built on an instance with $n = 7$ operations. The operation data are summarized below. At time instant 0, operation 2 is available and processed. At time instant 4, operation 1 becomes available, but operation 2 is more urgent ($q_2 > q_1$) and is consequently processed up

	1	2	3	4	5	6
r_i	4	0	9	15	20	21
p_i	6	8	4	5	8	8
q_i	20	25	30	9	14	16

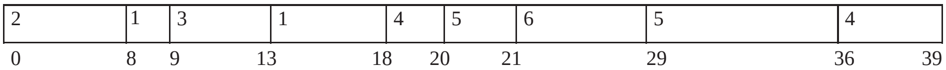


Fig. 1. JPS.

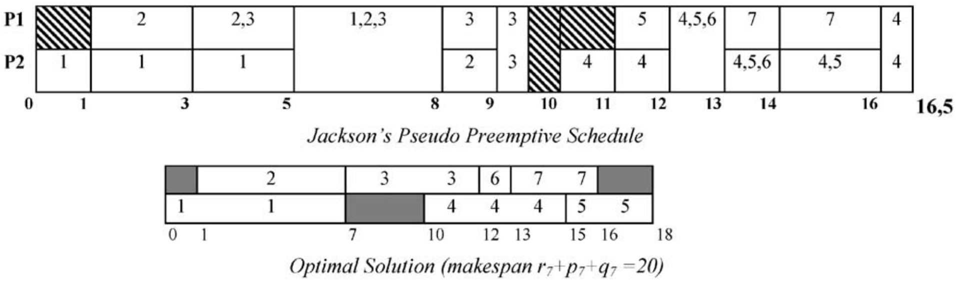


Fig. 2. JPPS and an optimal non-preemptive solution.

to its completion. At time instant 8, operation 1 starts, but is preempted by operation 3 at time 9, etc. The makespan of JPS for this instance is thus $\max_{i\in I} C_i + q_i = C_5 + q_5 = 50$.

2.3. JPPS

2.3.1. Informal description and notation

In this section, we define JPPS and we state some of its main properties. In a *pseudo-preemptive schedule*, the preemption of any available operation is also allowed, but we assume that a processor can be shared by a group of operations, and that an operation can be processed on more than one processor at a time. So, the number of processors assigned to an available operation i at time t, denoted by $\alpha_i(t)$, is not necessarily an integer. For building JPPS, we use a list algorithm whose priority dispatching rule is the complete tail $c_i(t) = q_i + a_i(t)$, where $a_i(t)$ is the remaining processing time of job i at the current time t in the list algorithm. So, contrarily to JPS, the priority attached to an available operation is not fixed over the time, but depends on its residual duration. The only restriction is that at any time t, we must have $a_i(t) \geqslant p_i - (t - r_i)$ for any operation i. An operation is said to be *partially available* if $a_i(t) = p_i - (t - r_i)$: such an operation can only be scheduled at a rate $\alpha_i(t) \leqslant 1$. Indeed, in this case, we have $p_i - a_i(t) = t - r_i$ and the part of operation i processed in time interval $[r_i; t]$ is as large as possible. It is said to be *totally available* if $a_i(t) > p_i - (t - r_i)$: such an operation can be processed at a rate $\alpha_i(t) \leqslant m$. Thus, JPPS schedules first the not in-process operations with maximal complete tail at a maximal rate consistent with their status (partially or totally available). JPPS is then composed of consecutive *schedule blocks* during which the subset of in-process operations and the associated rates are invariant, a schedule block B being partitioned into a set of partially available operations P and a set of totally available operations T. A schedule block starting at time t is completed at time $t + \theta$, called *decision time*, and associated with some event which leads to modifications on its structure. In such a block, operations of T are scheduled at the same rate α_T and those of P are processed at rate 1. The operations of the block are processed in $]t, t + \theta]$.

2.3.2. An example

Fig. 2 gives an example of JPPS built on an instance with $n = 7$ operations and $m = 2$ processors. The operation parameters are resumed in Table 1 below.

There are 12 blocks in this schedule. At time instant 0, operation 1 is the only available operation. So the first schedule block is $B = P = \{1\}$. At time instant 1, operations 1 and 2 are partially available, so $B = P = \{1, 2\}$, $T = \emptyset$. At time instant 2, operation 3 is available, but its priority is smaller than that of operations 1 and 2. At time instant 3, operations

Table 1
The data

	1	2	3	4	5	6	7
r_i	0	1	2	10	11	12	13
p_i	7	6	5	5	3	1	3
q_i	7	6	5	0	1	2	4

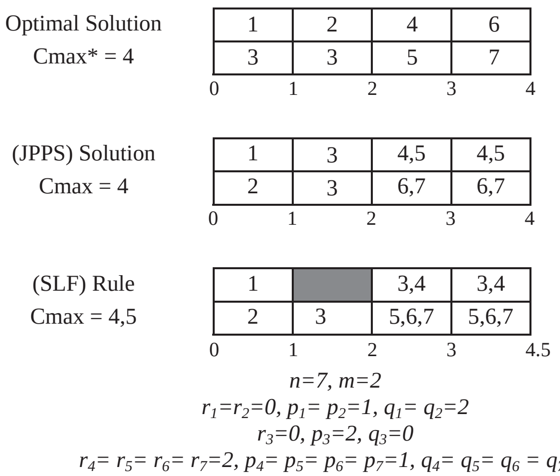


Fig. 3. First counter-example.

3 and 2 have now the same priority. So $B = P \cup T$, $P = \{1\}$, $T = \{2, 3\}$. At time instant 5, $B = T = \{1, 2, 3\}$, because the three operations have the same priority… . An optimal non-preemptive solution is also reported in Fig. 2. Its makespan is also equal to 20.

2.3.3. Two counter-examples

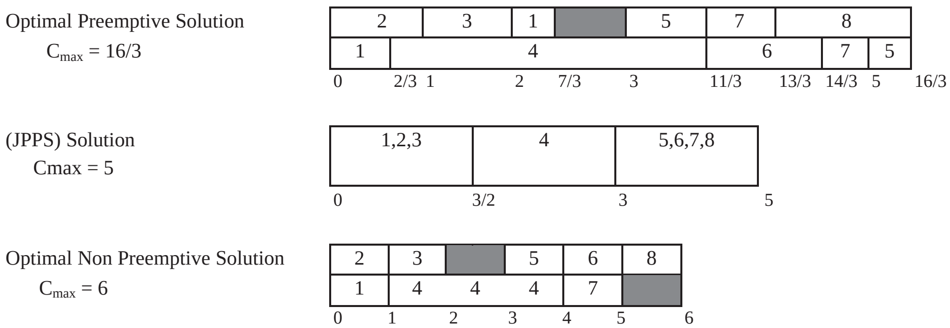
Note that if the operation rates are constrained to be less than or equal to 1 at any time t, it is possible to associate with the pseudo-preemptive schedule a preemptive schedule with the same makespan by building a McNaughton schedule on each schedule block. Unfortunately, the corresponding makespan is not systematically a lower bound for $\mathrm{Pm}/r_i,\ q_i/C_{\max}$. Fig. 3 presents a counter-example stating this fact. In [25], it is proved that compared with preemptive optimal schedule, SLF admits an absolute upper bound of $[(m-1)/m]\,p_{\max}$.

Lastly, Fig. 4 proposes an instance stating that JPPS does not systematically match the optimal preemptive solution. For this instance, getting a preemptive schedule with a makespan $C_{\max} = 5$ imposes that operations 1, 2 and 3 to be completed before time instant 2 ($q_1 = q_2 = q_3 = 3$). Consequently, only operation 4 can be processed in time slot (2,3), and in any preemptive schedule with makespan 5, this task will be executed during one time unit after $t = 3$. The effect is to shift operations 5, 6, 7, and 8 for at least 1/2 time unit right, which leads to a makespan greater than or equal to 5,5. It can be easily proved that the optimal preemptive schedule has a makespan of 16/3, whenever the optimal non-preemptive schedule has a makespan of 6.

2.3.4. Computing the decision times and schedule blocks

Two main steps condition the construction of JPPS: computing the decision times and computing the current schedule blocks.

Computing the current schedule block: As pointed out in Section 1, (JPPS) schedules first the not in-process available operations with maximal complete tail at a maximal rate consistent with their status (partially or totally available). In the sequel, A denotes the subset of available operations at time instant t. From [12], we have:



n=8, m=2

$r_1=r_2=r_3=0,\ p_1=p_2=p_3=1,\ q_1=q_2=q_3=3$

$r_4=0,\ p_4=3,\ q_4=0$

$r_5=r_6=r_7=r_8=3,\ p_5=p_6=p_7=p_8=1,\ q_5=q_6=q_7=q_8=0$

Fig. 4. Second counter-example.

Proposition 1 (Carlier and Pinson [12]). *Let t and t' denote two consecutive decision times in JPPS, and $B=P\cup T$ the schedule block starting at time t, where P is the set of partially available operations and T the set of totally available operations with maximal complete tail c_T. For any time $u\in\,]t;t']$, the operations of P are scheduled at rate* 1 *and the operations of T are scheduled at rate $\alpha_T=(m-|P|)/|T|$.*

Computing the decision times: It is easy to check that the events that can modify the current schedule block are of one of the following types:

(E_1) A not in-process operation becomes available.
(E_2) An in-process operation is completed.
(E_3) A not in-process available operation enters into the process.
(E_4) A totally available operation becomes partially available.
(E_5) A partially available operation becomes totally available.

At each step of the algorithm, the next decision time is computed according to these five possible events. More precisely, assume that we have built JPPS up to time t. The related schedule block is invariant up to the first time $t+\theta$ where an event of one of the five types defined above occurs. From [12], we have,

Proposition 2 (Carlier and Pinson [12]). *$\theta=\min(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5)$ where $t+\theta_k$, $k\in[1,5]$, is the first time instant at which an event of type E_k can occur, and*

$$\theta_1=\min_{j\in\bar{A}} r_j-t,\quad \theta_2=\min_{j\in B}\frac{a_j(t)}{\alpha_j(t)},\quad \theta_3=\min_{j\in B}\frac{c_j(t)-c_{\max}}{\alpha_j(t)},\quad \theta_4=\min_{j\in T}\frac{t-[r_j+p_j-a_j(t)]}{\alpha_T-1},\quad \theta_5=\min_{j\in P}\frac{c_j(t)-c_T}{(1-\alpha_T)},$$

with $c_{\max}=\max_{i\in A\setminus B}c_i(t)$, and c_T is the maximal complete tail of operations in T.

Proposition 3 (Carlier and Pinson [12]).

- *The maximal number of events of type* E_1–E_3 *is* $O(n)$.
- *The maximal number of events of type* E_4 *and* E_5 *is* $O(nm)$.

A direct corollary of this proposition is

Theorem 1 (Carlier and Pinson [12]). *The maximal number of schedule blocks in JPPS is* $O(nm)$.

2.3.5. *Basic properties of JPPS*

Let C_j denote the completion time of operation j in JPPS. By convention, we set $a_j(t)=-\infty$ for $t>C_j$. So we have: $C(JPPS)=\max_{j\in I,t\in IR}(t+a_j(t)+q_j)$. We also have the following result:

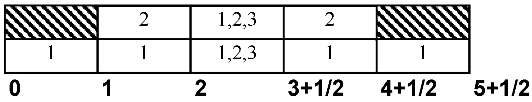


Fig. 5. Instance with $m=2$ and $n=3$.

Theorem 2 (Carlier and Pinson [12]).

$$C(\mathrm{JPPS}) = \max\left\{\max_{i\in I}(r_i + p_i + q_i), \quad \max_{J\subseteq I, |J|\geqslant m} G'(J)\right\}.$$

J denoting a subset of operations of I with $|J|\geqslant m$, and $G'(J)$ the quantity defined by,

$$G'(J) = \frac{1}{m}(r_{i_1} + r_{i_2} + \cdots + r_{i_m}) + \frac{1}{m}\sum_{i\in J} p_i + \frac{1}{m}(q_{j_1} + q_{j_2} + \cdots + q_{j_m}),$$

where $i_1, i_2 \dots i_m$ (resp. $j_1, j_2, \dots j_m$) denote the m first jobs in J rearranged in an ascending order of heads (resp. tails).

Theorem 3 (Carlier and Pinson [12]). *C (JPPS) can be computed in $O(n\log n + nm\log m)$ time.*

3. Structure of JPPS: further results

3.1. Introduction

This section is dedicated to new results related to the structure of JPPS. First, we state that there exist instances of JPPS for which the number of operation parts are, respectively, $O(n^2)$ and $O(nm^2)$. Next, we propose an algorithm computing explicitly JPPS in time $O(n^2 + nm^2)$. Lastly, we show that JPS and JPPS have similar structures by focusing on their respective schedule block composition.

3.2. Number of schedule blocks and operation parts in JPPS: worst case analysis

As stated in [12], the number of consecutive blocks in JPPS is $O(nm)$ and the number of operation parts involved in these blocks is $O(n^2 + nm^2)$. An interesting question is: may these upper bounds be reached in practice? The answer is yes, as shown in the following result:

Proposition 4. *There exist instances of JPPS for which the number of operation parts are, respectively, $O(n^2)$ and $O(nm^2)$.*

Proof. It is very easy to build an instance with $O(n^2)$ parts in JPPS. For example, it occurs when the operations are nested. The following data match this property:

$$r_1 = 0, \quad r_k = r_{k-1} + 1, \quad k = 2, \dots, m,$$

$$r_k = r_{k-1} + (k-1)/m, \quad k = m+1, \dots, n,$$

$$p_1 = 2n - 1, \quad p_k = p_{k-1} - 2, \quad k = 2, \dots, n,$$

$$q_1 = 0, \quad q_k = q_{k-1} + 1, \quad k = 2, \dots, n.$$

The data are chosen in such a way that when an operation becomes available, it has exactly the same priority than other available operations. So, it enters the current schedule block. It is easy to check that for such an instance, the number of operation parts involved in JPPS is $O(n^2)$. Fig. 5 gives the Gantt chart associated with the instance with $m=2$ and $n=3$.

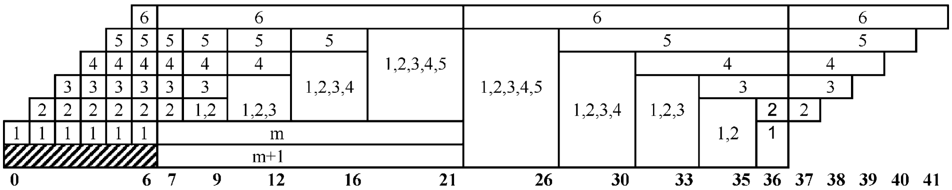


Fig. 6. Instance with $m = 7$ and $t = 1$.

It is more difficult to build an instance with $O(nm^2)$ operation parts in JPPS. We propose the following instance matching this goal (t denotes a predefined parameter):

- the first group contains $m-1$ operations:

$$r_1 = 0, \quad r_2 = 1, \ldots, r_{m-1} = m-2,$$

$$p_1 = p_2 \ldots p_{m-1} = P \quad (P \text{ sufficiently large, e.g. } P = (m-1)(m-2)t + (m-1)),$$

$$q_1 = q_2 = \cdots = q_{m-1} = 0,$$

- the second group contains $2t$ operations:

$$p_{m+i-1} = p_{(m+i-1)+t} = (m-1)(m-2)/2 \quad (i = 1, \ldots, t),$$

$$r_{m+i-1} = r_{(m+i-1)+t} = (m-1) + (i-1)(m-1)(m-2) \quad (i = 1, \ldots, t),$$

$$q_{m+i-1} = q_{(m+i-1)+t} = Q \quad (i = 1, \ldots, t)$$

with Q sufficiently large, e.g. $Q = (t+1)(m-1)(m-2)$.

Fig. 6 reports the Gantt chart associated with JPPS for the corresponding instance with $m = 7$ and $t = 1$.

The ideas underlying this construction are the following ones: the operations of the first group have smaller priorities than the operations of the second group. Two operations of the second group are available at times $(m-1), (m-1)+(m-1)(m-2), \ldots, (m-1)+(t-1)(m-1)(m-2)$. They are processed at rate 1, during $(m-1)(m-2)/2$ time units. All the operations of the first group are processed during $(m-1)(m-2)$ time units in each of the time intervals $[(m-1), (m-1)+(m-1)(m-2)]$, $[(m-1)+(m-1)(m-2), (m-1)+2(m-1)(m-2)], \ldots, [(m-1)+(t-1)(m-1)(m-2), (m-1)+(t)(m-1)(m-2)]$. They are partially available at time $(m-1), (m-1)+(m-1)(m-2), \ldots, (m-1)+(t-1)(m-1)(m-2)$. Each operation of the first group is splitted into $m-2$ parts in each of the previous interval. So it is splitted into at least $t(m-2)$ parts. Because there are exactly $m-1$ operations in the first group and $t = O(n)$, the total number of parts is $O(nm^2)$. □

A practical consequence of this result is that the explicit construction of JPPS may also require a similar computational effort. For this reason, we propose in the next section an algorithm dealing with the explicit building of JPPS with a complexity $O(n^2 + nm^2)$, and leading to a much easier implementation than the one proposed in our previous paper [12].

3.3. Computing JPPS in $O(n^2 + nm^2)$

A simple data structure: An explicit construction of JPPS can be performed in an efficient way by using a data structure having two levels of chained lists (cf. Fig. 7). The first list chains groups of available operations in a decreasing order of their priorities (complete tails). Operations in a same group are then chained in a decreasing order of their residual processing times (i.e: q_i increasing).

The algorithm: At each step of the construction of JPPS (decision time), we first scan the head group which has a maximal priority in order to determine the status of its operations (partially or totally available) and to determine the current set of totally available operations T. Next, we compute the processing rate for the operations of this group. If some processor remains idle, the same process is performed with the second group of operations and so on. Then, the next decision time is computed using Proposition 2, and the residual processing times of the operations involved in the current schedule block are updated. A completed operation is removed from its associated list.

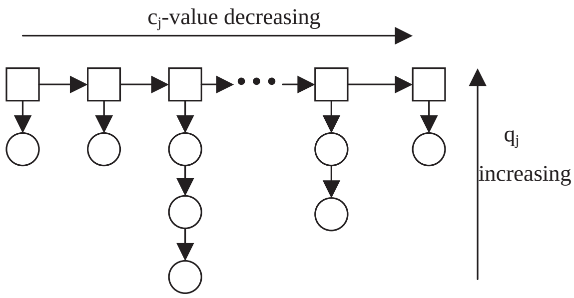


Fig. 7. Data structure.

Proposition 5. *The explicit construction of JPPS using the algorithm above is performed in time* $O(n^2) + O(nm^2)$.

Proof. Let us examine the complexity associated with the different events conditioning the construction of JPPS. An event E_1 occurs when an operation becomes available. Clearly, positioning this operation in the structure using the double key (priority, residual processing time) requires only $O(n)$ time. There are n events E_1, so the associated overall complexity is $O(n^2)$. An event E_2 occurs when an operation is completed. It can be tested in $O(m)$ because this operation has a minimal residual processing time in the current schedule block, and the number of groups in process is $O(m)$. Then removing the operation from the structure and restoring it latter can be performed in constant time. It costs globally $O(nm)$. An event E_3 occurs when a not in-process available operation enters into the process, and $O(n)$ times. The merging of groups of operations with equal priority can be costly. Indeed, lists can be of size $O(n)$, $O(n)$ times, and the overall complexity is $O(n^2)$. Otherwise, they are of size $O(m)$, which implies a global cost of $O(n^2m)$. An event E_4 occurs when a totally available operation becomes partially available, and $O(nm)$ times. Since $|T| < m$ in this case, the overall complexity associated with this event is $O(nm^2)$. Last, an event E_5 occurs when a partially available operation becomes totally available, and at most $O(nm)$ times. As pointed out in [12], such an event involving more than $2m$ operations cannot occur more than n times, and the associated global complexity is $O(n^2 + nm^2)$. That completes the proof. □

3.4. Component structures of JPPS

As stated in [11], JPS has a very nice structure relying on an enhanced notion of schedule component and defined below.

Definition 1. The component K_c associated with any operation c in JPS is the maximal set of tasks (for the inclusion) satisfying:

- $c \in K_c$,
- $q_c = \min_{j \in K_c} q_j$,
- $C_c = \min_{j \in K_c} r_j + \sum_{j \in K_c} p_j$.

It is computed by starting from C_c and backwarding down to the first time t where either the machine is idle or there is an operation processed in $[t-1, t[$ with a smaller priority than c. Components are of prime interest because of the following property:

Proposition 6 (Carlier and Pinson [12])**.**

- *in JPS, components are either included or disjoined*: $\forall (i,j) \in I^2,\ K_i \cap K_j = \emptyset \vee K_i \subset K_j \vee K_j \subset K_i$,
- $C(JPS) = \max_{j \in I} (C_j + q_j) = \max_{j \in I} h(K_j)$.

For the instance depicted in Fig. 1, we obtain

- $K_1 = \{1, 2, 3\}$, $K_2 = \{2\}$, The related global structure is the following one:
- $K_3 = \{3\}$, $K_4 = \{1, 2, 3, 4, 5, 6\}$,

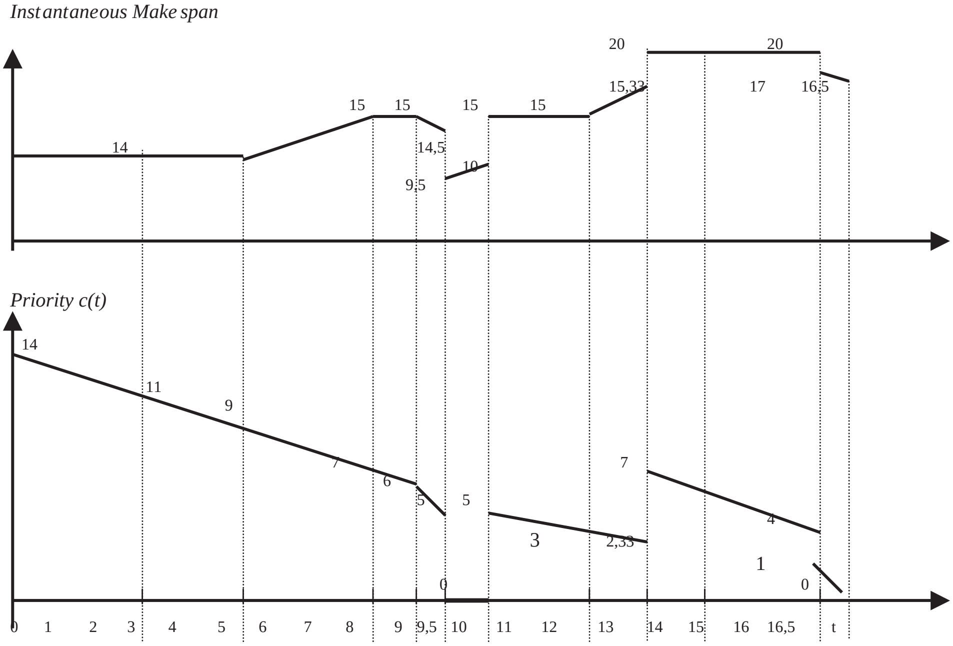
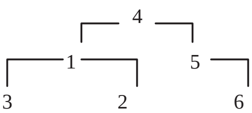


Fig. 8. The instantaneous makespan.

- $K_5 = \{5, 6\}$, $K_6 = \{6\}$.



Unfortunately, the structure of JPPS is not so simple. Nevertheless, some interesting properties remain valid.

Definition 2 (Instantaneous priority and makespan)**.** Let us associate with JPPS the instantaneous priority $c(t)$, defined, at time t, as the maximal complete tail of any task in process at t:

$$c(t) = \max[c_T, \max_{i \in P(t)} c_i(t)].$$

$c(t)$ is a piecewise linear function, with a finite number of discontinuity points corresponding to arrivals or departures of operations.

The instantaneous makespan associated with time instant t is then defined by

$$C_{\max}(t) = t + c(t).$$

Fig. 8 reports the functions $c(t)$ and $C_{\max}(t)$ for the instance presented in Section 2.3.2.

From this example, we can notice that a strict increase of the makespan only occurs for time instants where $\alpha_T < 1$. Moreover, local optima for the makespan are located on completion times of operations, similarly to JPS.

Component structure of JPPS

Definition 3 (renewable time (r-time))**.** A decision time $\tau \in [0, C(\text{JPPS})]$ is a renewable time or r-time if, for any $\varepsilon > 0$ sufficiently small:

(C1) at least one processor is idle at $\tau - \varepsilon$, or
(C2) some operation with a priority strictly smaller than those processed at time $\tau + \varepsilon$ are scheduled at $\tau - \varepsilon$,
(C3) all processors are busy at time $\tau + \varepsilon$.

We have:

Lemma 1. *Let B be the block starting at a renewable time τ. B contains at least m operations. Moreover, operations of B are partially available at τ, and there exists an operation $c \in B$ such that $\tau = r_c$.*

Proof. Let i be an operation of B and let us suppose that $r_i < \tau$ (if $r_i = \tau$, then i is necessarily partially available). If i was totally available at τ, i would have been executed in the block B' preceding block B, because of its priority, or because some processor is idle at $\tau - \varepsilon$. So, any operation in B is partially available at τ. If B contains stricly less than m operations, then some processor should be idle at time $\tau + \varepsilon$, which leads to a contradiction because of (C3). Last, because of (C1) or (C2), an operation c in B becomes available at τ and $\tau = r_c$. □

Now, let $t \in [0, C(\text{JPPS})]$ be a time instant at which no processor is available. We define the renewable time associated with t, $\tau(t)$, as the largest r-time preceding t satisfying $c(\tau(t)) < c(t)$. Let $J(t)$ be the set of operations having some parts scheduled in $]\tau(t), t]$.

Lemma 2. *Let $i_1, i_2, \ldots, i_m$ be the operations of $J(t)$ having the smallest release dates*:

$$mt = r_{i1} + r_{i2} + \cdots + r_{im} + \sum_{i \in J(t)} (p_i - a_i(t)).$$

Proof. From Lemma 1, all the operations in-process just after $\tau(t)$ are partially available at $\tau(t)$. If another operation of $J(t)$ was available before $\tau(t)$, it would be processed in the block associated with $]\tau(t) - \varepsilon,\ \tau(t)]$. Consequently, we have

$$\tau(t) = r_i + p_i - a_i(\tau(t)) \quad \forall i \in B,$$

and we obtain

$$m\tau = r_{i1} + r_{i2} + \cdots + r_{im} + \sum_{i \in J(t)} (p_i - a_i(\tau(t))).$$

Moreover, all the processors being busy between $\tau(t)$ and t, we have

$$mt = m\tau(t) + \sum_{i \in J(t)} (a_i(\tau(t)) - a_i(t)).$$

The claimed formula is then obtained by a simple substitution. □

Remark. By taking $t = C_j$, we get, if $T(C_j) \neq \emptyset : C_j = (1/m)[r_{i1} + r_{i2} + \cdots + r_{im} + \sum_{i \in J(Cj)} (p_i - a_i(C_j))]$,

which is similar to the one machine case.

Lemma 3. *Let t_1 and t_2 be two time instants of the schedule horizon of JPPS. One of the three following cases occurs*:

(1) $]\tau(t_1), t_1[\ \subseteq\ ]\tau(t_2), t_2[$,
(2) $]\tau(t_2), t_2[\ \subseteq\ ]\tau(t_1), t_1[$,
(3) $]\tau(t_1), t_1[\ \cap\ ]\tau(t_2), t_2[\ = \emptyset$.

Proof. This is true by definition of $\tau(t)$. □

Lower bound of the makespan

Definition 4. t_o is a locally maximal point if:

- $C_{\max}(t_o + \varepsilon) < C_{\max}(t_o)$,
- $C_{\max}(t_o) \geqslant C_{\max}(t_o - \varepsilon)$

for ε sufficiently small.

Lemma 4. *Locally maximal points are located on completion times of the operations.*

Proof. Let t_o be a locally maximal point, and let $B = P(t_o) \cup T(t_o)$ denote the block ending at t_o. If $P(t_o)$ is not empty, then we have: $C_{\max}(t_o) = r_{io} + p_{io} + q_{io}$, where i_o denotes the operation of $P(t_o)$ with maximal tail. Otherwise, $P(t_o)$ is empty and $B = T(t_o)$. This block contains at least m operations, processed at a rate $\leqslant 1$, because $C_{\max}(t_o) \geqslant C_{\max}(t_o - \varepsilon)$. Let B' be the block starting at t_o. B' can eventually be empty or it contains strictly less than m operations, because $C_{\max}(t_o + \varepsilon) < C_{\max}(t_o)$. Consequently, some operation $j \in B$ is completed at t_o and $t_o = C_j$. □

Proposition 7. *Let θ be a locally maximal point corresponding to a completion time of an operation $j(C_j=\theta)$. Moreover, let us suppose that j is totally available at θ and that $P(\theta)=\emptyset$. We have*: $G'(J(\theta)) = C_j + q_j = C_{\max}(\theta)$.

Proof. Indeed, in the block ending at θ, there are more than m operations because $C_{\max}(\theta)$ is supposed to be locally maximal $(C_{\max}(\theta - \varepsilon) \leqslant C_{\max}(\theta))$. Moreover, there are less than m operations in process at $\theta + \varepsilon$ which are not completed at $\theta(C_{\max}(\theta) > C_{\max}(\theta + \varepsilon))$, and $C_i = a_i(\theta) + q_i = q_j(i \in B)$. We get the result by using Lemma 2. □

From what preceeds, and similarly to JPS, we can associate with any operation j satisfying the condition of Proposition 7 a component K_j defined by $K_j = J(C_j)$. As pointed out in the previous remark, we have: $C_j = (1/m)[r_{i1} + r_{i2} + \cdots + r_{im} + \sum_{i \in J(Cj)} (p_i - a_i(C_j))]$, and we can claim the following result:

Theorem 4.

- *in JPPS, components are either included or disjoined*: $\forall (i,j) \in I'^2$, $K_i \cap K_j = \emptyset \vee K_i \subset K_j \vee K_j \subset K_i$
- $C(\text{JPPS}) = \max_{j \in I}(C_j + q_j) = \max[\max_{j \in I}(r_j + p_j + q_j),\ \max_{j \in I'} G'(K_j)]$, *where I' corresponds to the set of operations totally available at their completion times in JPPS.*

Discussion: The points matching local maximality of the instantaneous makespan correspond to completion times of operations. These points can be associated with critical subsets J or with single operations. So they are very interesting for bounding the makespan. Moreover, these points are obtained without additional cost in $O(n \log n + nm \log m)$. It is sufficient to store the generalized makespan $C_{\max}(t)$ and the priority $c(t)$ for the consecutive decision times to get the interesting values.

4. Adjustment of heads

4.1. Introduction

In Carlier and Pinson [10], we proposed the following algorithm for adjusting heads in $O(n^2)$ for the $1/r_i,\ q_i/C_{\max}$ problem. In the sequel, UB is assumed to be an upper bound of the optimal makespan (heuristic solution for instance).

- Build JPS up to r_c.
- Take the operations of $K_c^+ = \{j \in I / a_j(r_c) > 0\}$ in the increasing order of the tails and find the first one s such that

$$r_c + p_c + \sum_{\{j \in K_c^+ / q_j \geqslant q_s\}} a_j(r_c) + q_s > \text{UB} \quad \text{and} \quad q_s > q_c \text{ (if any exists)}.$$

- Define $K_c^* = \{j \in K_c^+ / q_j \geqslant q_s\}$.
- Adjust r_c by setting: $r_c = \alpha_c = \max_{j \in K_{c^*}} C_j$.

Indeed, the idea is to build JPS under the constraint that operation c is processed between r_c and $r_c + p_c$ and to check whether or not the resulting schedule has a makespan strictly larger than UB. If so, we can increase the release date of c. We show in the next section that a similar strategy can be performed for the $\text{P}m/r_i,\ q_i/C_{\max}$ scheduling problem using JPPS. However, it is more complex since p_c has to be replaced by a part of it, and there are m operations to be considered instead of one.

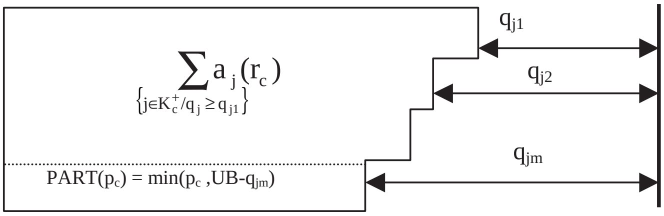


Fig. 9. Adjusting heads.

4.2. *Adjustments of heads for* $\mathrm{Pm}/r_i, q_i/C_{\max}$

Let us assume that operation c has to be executed in JPPS between r_c and $r_c + p_c$, that is c is supposed to have an infinite priority. We now have the formula

$$(1/m)[mr_c + \mathrm{PART}(p_c) + \sum_{\{j\in K_c^+/q_j \geqslant q_{j_1}\}} a_j(r_c) + q_{j1} + q_{j2} + \cdots + q_{jm}] > \mathrm{UB}, \tag{1}$$

where $\mathrm{PART}(p_c) = \min(p_c, \mathrm{UB} - q_{jm})$ and $K_c^+ = \{j/a_j(r_c) > 0\}$ (cf. Fig. 9).

Eq. (1) can be rewritten as

$$(1/m)[mr_c + \mathrm{PART}(p_c) + \sum_{\{j\in K_c^+/q_j > q_{j_m}\}} a_j(r_c) + (a_{j1}(r_c) + q_{j1}) + (a_{j2}(r_c) + q_{j2}) + \cdots + (a_{jm}(r_c) + q_{jm})] > \mathrm{UB}, \tag{2}$$

or

$$(1/m)[mr_c + \mathrm{PART}(p_c) + \sum_{\{j\in K_c^+/q_j > q_{j_m}\}} a_j(r_c) + c_{j1}(r_c) + c_{j2}(r_c) + \cdots + c_{jm-1}(r_c) + (a_{jm}(r_c) + q_{jm})] > \mathrm{UB}.$$

Let us set

$$\eta(q_{jm}) = \sum_{\{j\in K_c^+/q_j > q_{jm}\}} a_j(r_c)$$

$$v(q_{jm}) = \max_J\,[c_{j1}(r_c) + c_{j2}(r_c) + \cdots + c_{jm-1}(r_c)]$$

with

$$J = \{(j_1, j_2, \ldots, j_{m-1}) \in (K_c^+)^{m-1}/q_{j1} < q_{jm}, q_{j2} < q_{jm}, \ldots, q_{jm-1} < q_{jm}\}$$

and

$$L(q_{jm}) = r_c + (1/m)\mathrm{PART}(p_c) + (1/m)\eta(q_{jm}) + (1/m)v(q_{jm}) + (1/m)(a_{jm}(r_c) + q_{jm}). \tag{4}$$

The problem is then to find an operation j_m (if any exists) such that $L(q_{jm}) > \mathrm{UB}$. In this case, we have to adjust r_c to attempt a feasibility recovering.

From (4), we obtain the adjustment: $r_c \leftarrow r_c + (L(q_{jm}) - \mathrm{UB}) \times m$.

For a computational point of view, we state the following result:

Proposition 8. *Adjustments of heads for the* Pm/r_i, $q_i/C_{\max}$ *scheduling problem can be performed using JPPS in time* $\mathrm{O}(n^2)$.

Proof. Using the data structure depicted in Section 3, it is easy to see that, once JPPS has been built up to time instant r_c, the determination of operation j_m, if any exists, can be computed in time $\mathrm{O}(n)$ by iterative adjustments of both $\eta(q_{jm})$ and $v(q_{jm})$-values (cf. Fig. 10):

Without loss of generality, we can assume that all tails are distinct (if $q_i = q_j$, then we simply set $q_i = q_j + \varepsilon$). Let us denote by $J_m^+ = \{j\in K_c^+/q_j > q_{jm}\}$, $J_m^- = \{j\in K_c^+/q_j < q_{jm}\}$, and $K_u = \{j\in K_c^+/c_j(r_c) = \sigma_u\}$ $(u\in]s])$, where σ_1, $\sigma_2, \ldots, \sigma_s$ are the s distinct $c_j(r_c)$-values over the operations in K_c^+. Updating $\eta(q_{jm}) = \sum_{j\in J_m^+} a_j(r_c)$ for two consecutive values of q_{jm} can simply be done in constant time if tails are sorted in decreasing order. Now, once q_{jm} is set, the computation of $v(q_{jm})$ (under the constraints: $q_{j1} < q_{jm}, q_{j2} < q_{jm}, \ldots, q_{jm-1} < q_{jm}$) can be performed by scanning sets $K_sJ_m^-$, $K_s \cap J_m^-, \ldots, K_1 \cap J_m^-$, in order to get the first $m-1$ operations giving to $[c_{j1}(r_c) + c_{j2}(r_c) + \cdots + c_{jm-1}(r_c)]$ a

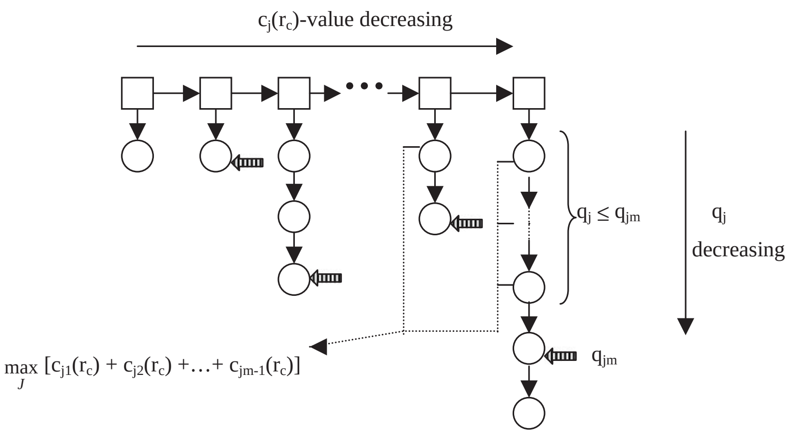


Fig. 10. The data structure.

maximal value. Since for two consecutive and decreasing values of q_{jm}, only one set $K_u \cap J_m^-$ is updated, adjusting $v(q_{jm})$ can also be performed in amortized constant time. Thus, adjustments of $\eta(q_{jm})$ and $v(q_{jm})$ for consecutive and decreasing values of q_{jm} can be performed in O(n) time. Consequently, the overall complexity associated with the adjustments of all heads is O(n^2). □

The adjustments described above can be enhanced using the following property (the same notation is used).

Proposition 9. *If*

- $r_c + p_c + c_{j1}(r_c) > \mathrm{UB}, \ldots, r_c + p_c + c_{jm}(r_c) > \mathrm{UB}$,
- $r_c + (1/m-1)\eta(q_{jm}) + (1/m-1)v(q_{jm}) > \mathrm{UB}$,

then $r_c \geqslant \min_{\{j \in K_c^+/q_j \geqslant q_{j1}\}} (r_j + p_j)$.

Proof. Operation c cannot be processed before any operation j_k on the same processor. Since $r_c + (1/m-1)\eta(q_{jm}) + (1/m-1)v(q_{jm}) > \mathrm{UB}$, operations of $J = \{j/q_j \geqslant q_{j1}\}$ cannot be executed on $m-1$ processors with a makespan less than or equal to UB. The related adjustment of r_c relies on the immediate selection principle proposed in [11] for the $1/r_i$, $q_i/C_{\max}$ scheduling problem. □

Notice that the application of this result together with JPPS based adjustments described above does not require any additional computational effort.

Discussion: These adjustments can be improved in two ways. The first one consists in considering at time r_c the partially available operation d with maximal residual processing time. It is quite immediate to adapt the approach in O(n^2). The second way was proposed by Baptiste et al. in [2], but is too costly for a practical use. Of course, a similar technique can be used for adjusting tails.

5. Lower bound and adjustments of heads for the CuSP using JPPS

As pointed out in Section 1, the CuSP is a generalization of the Pm/r_i, $q_i/C_{\max}$ scheduling problem, where each activity requires a constant amount e_i of resource throughout its processing. In this section, we show that JPPS can be used for bounding the makespan as well as for adjusting heads and tails for the CuSP with similar complexities.

A first idea for treating this general case is to associate with each operation i e_i operations requiring one machine, and then to apply JPPS on the derived instance involving $\sum e_i$ operations. Clearly, such a strategy introduces a pseudo-polynomial component in the complexity associated with JPPS construction. However, in the corresponding schedule, each operation derived from i is executed during the same periods with the same rates. So, a second idea is to modify the rules defining rates in the following way:

$$\alpha_i(t) = e_i \quad \text{if } i \in P,$$

$$\alpha_i(t) = \frac{m - \sum_{i \in P} e_i}{\sum_{i \in T} e_i} \times e_i.$$

Consequently, JPPS can be computed for the CuSP with the same complexity than the $\mathrm{Pm}/r_i,\ q_i/C_{\max}$ scheduling problem ($e_i = 1, \forall i \in I$).

For the CuSP, each operation i is replaced implicitly by e_i operations as explained before. Thus, the process described in Section 4.2 can simply be transposed to the CuSP, with the same complexity. A second idea is to associate parallel machine problems with a CuSP instance. Indeed, if we restrict the problem to operations such that $e_i > m/(k+1)$, we get a k machine problem instance. This technique is probably powerful for $k = 1$ and 2 [6,8].

6. Conclusion

This paper presents new results on the structure of JPPS applied to the $\mathrm{Pm}/r_i,\ q_i/C_{\max}$ scheduling problem. This particular schedule generalizes JPS for $1/r_i,\ q_i/C_{\max}$, and has numerous similarities with the latter as pointed out in this study. The complexity associated with their respective makespan computations are nearly the same as well as their worst case performance ratio. Moreover, the list schedules associated with both JPS and JPPS have very closed structures. In addition, adjustments of heads and tails can be performed for the $\mathrm{Pm}/r_i,\ q_i/C_{\max}$ scheduling problem using a strategy very close to the one designed in [12] for the $1/r_i,\ q_i/C_{\max}$ scheduling problem. Furthermore, we propose a simple adaptation of these tools (lower bounds and adjustments of heads) for the CuSP without any additional computational effort. Lastly, such techniques can be used in implicit enumerative methods for solving to optimality m processor scheduling problems as well as the RCPSP.

Acknowledgements

We acknowledge Philippe Baptiste and Emmanuel Néron for their fruitful advice, and anonymous referees for their constructive comments.

References

[1] E. Balas, Machine sequencing via disjunctive graphs: an implicit enumeration algorithm, Oper. Res. 17 (1969) 941–957.
[2] P. Baptiste, C. Le Pape, W. Nuijten, Satifiability tests and time-bound adjustments for cumulative scheduling problems, Ann. Oper. Res. 92 (1999) 305–333.
[3] P. Brucker, B. Jurisch, A. Krämer, The job-shop and immediate selections, Ann. Oper. Res. 50 (1992) 93–114.
[4] P. Brucker, B. Jurisch, B. Sievers, A branch & bound algorithm for the job-shop scheduling problem, Discrete Appl. Math. 49 (1994) 109–127.
[5] J. Carlier, The one-machine sequencing problem, European J. Oper. Res. 11 (1982) 42–47.
[6] J. Carlier, Problèmes d'ordonnancements á contraintes de ressources: algorithmes et complexité, Thèse d'état, Université Paris VI, 1984.
[7] J. Carlier, Scheduling jobs with release dates and tails on identical machines to minimize makespan, European J. Oper. Res. 29 (1987) 298–306.
[8] J. Carlier, B. Latapie, Une méthode arborescente pour résoudre les problèmes cumulatifs, RAIRO 25 (3) (1991) 311–340.
[9] J. Carlier, E. Pinson, An algorithm for solving the job shop problem, Management Sci. 35 (1989) 164–176.
[10] J. Carlier, E. Pinson, A practical use of Jackson's preemptive schedule for solving the job-shop problem, Ann. Oper. Res. 26 (1991) 269–287.
[11] J. Carlier, E. Pinson, Adjusting heads and tails for the job-shop problem, European J. Oper. Res. 78 (1994) 146–161.
[12] J. Carlier, E. Pinson, Jackson's pseudo preemptive schedule for the $\mathrm{Pm}/r_i,\ q_i/C_{\max}$ scheduling problem, Ann. Oper. Res. 83 (1998) 41–58.
[13] E.G. Coffman, R.R. Muntz, Preemptive scheduling on two-processor systems, IEEE Trans. Comput. C-18 (1970) 1014–1020.
[14] E. Demeulemeester, W. Herroelen, A branch and bound procedure for the multiple resource-constrained project scheduling problem, Management Sci. 38 (1992) 1803–1818.
[15] E. Demeulemeester, W. Herroelen, Recent advances in branch and bound procedures for resource-constrained project scheduling problems, in: P. Chretienne, et al., (Eds.), Scheduling Theory and Its Applications, Wiley, New York, 1995.

[16] H. Hoogeven, C. Hurkens, J.K. Lenstra, A. Vandevelde, Lower bounds for the multiprocessor flow shop, Second Workshop on Models and Algorithms for Planning and Scheduling, Wernigerode, May 22–26, 1995.
[17] W. Horn, Some simple scheduling algorithms, Naval. Res. Logist. Quart. 21 (1974) 177–185.
[18] J.R. Jackson, Scheduling a production line to minimize maximum tardiness, Research Report 43, Management Science Research Project, University of California, Los Angeles, 1955.
[19] J. Labetoulle, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Preemptive scheduling of uniform machines subject to release dates, Progress in Combinatorial Optimization, Academic Press, Florida, 1984, pp. 245–261.
[20] B.J. Lageweg, J.K. Lenstra, A.H.G. Rinnooy Kan, Minimizing maximum lateness on one machine: computational experience and some applications, Statist. Neerlandica 30 (1976) 25–41.
[21] E.L. Lawler, Preemptive scheduling of precedence constrainted jobs on parallel machines, in: Dempster et al. (Eds.), Deterministic and Stochastic Scheduling, Reidel, Dordrecht, 1982, pp. 101–123.
[22] Z. Liu, E. Sanlaville, Profile scheduling by list algorithms, in: P. Chretienne, et al., (Eds.), Scheduling Theory and Its Applications, Wiley, New York, 1995.
[23] E. Néron, Du flow-shop hybride au problème cumulatif, Thèse de l'Université de Technologie de Compiégne, 1999.
[24] E. Pinson, The job shop scheduling problem : a concise survey and recent developments, in: P. Chretienne, et al., (Eds.), Scheduling Theory and Its Applications, Wiley, New York, 1995.
[25] E. Sanlaville, Nearly on line scheduling of preemptive independent tasks, Discrete Appl. Math. 57 (1995) 229–241.